`socklab v6`
Laboratory of Internet *sockets*
User Guide
Reference Guide

Amine *BOUKIOUD*
Quang − Ngoc *DUONG*
Rafik *NASRI*

Date : 2011 − 06 − 06(*Monday*, *june* 6*th* 2011)

Contents

1 Introduction **5**

2 General presentation **6**
2.1 Classification of commands 6
2.2 Operating Modes of `socklab v6` 6
2.3 Principle of use 7
2.4 Launch `socklab v6` 7

3 Example of using `socklab v6` **8**

4 `socklab v6`'s Reference Guide **12**
4.1 Utilization 12
4.2 Special commands 12
4.2.1 `quit`—`exit` 13
4.2.2 `help` 13
4.2.3 `status` 13
4.3 Standard Mode Commands 13
4.3.1 `socket` 13
4.3.2 `bind` 14
4.3.3 `listen` 14
4.3.4 `accept` 14
4.3.5 `connect` 14
4.3.6 `close` 14
4.3.7 `shutdown` 15
4.3.8 `write` 15
4.3.9 `read` 15
4.3.10 `send` 15
4.3.11 `recv` 16
4.3.12 `sendto` 16
4.3.13 `recvfrom` 16
4.3.14 `options` 17
4.4 Commands in TCP mode 17
4.4.1 `passive` 17
4.4.2 `accept` 17
4.4.3 `connect` 17
4.4.4 `close` 18

4.4.5 `shutdown` . . . 18
4.4.6 `write` . . . 18
4.4.7 `read` . . . 18
4.4.8 `usend` . . . 18
4.4.9 `urecv` . . . 18
4.4.10 `options` . . . 18
4.5 Commands in UDP mode . . . 19
4.5.1 `socket` . . . 19
4.5.2 `close` . . . 19
4.5.3 `sendto` . . . 19
4.5.4 `recvfrom` . . . 19
4.5.5 `options` . . . 19

Chapter 1

Introduction

Within a same machine, processes have some ways to connect between them: files, pipes, signals, messages ... but these processes must be executed on the same machine.

Two processes execute in two different machines can only communicate each other if these machines are connected. But they have to use a programming interface: it is a set of primitives whose purpose is to relate the processes in two machines.

There are many network programming interfaces in UNIX systems ; the most currently used is *sockets* BSD (*Berkeley Software Distribution*). First, the socket term is used to designate access points that processes must handle to send or receive data. By extension, this term refers to any programming interface (known as sockets).

Using this API is not easy: it is rare that two processes (even located on the same machine) arrive at the first attempt to communicate via sockets. Indeed, the primitives are generally not used, or misused data structures, so that the first attempt is frequently unsuccessful (not mention the errors that occur directly when compiling programs).

`socklab v6` is a utility that allows you to manipulate sockets, without having to compile a single line of code. This laboratory study of sockets is in the form of shell, where each command is associated with the call of a primitive interface (sometimes many primitives). Whenever you enter a command, `socklab v6` runs the corresponding original and gives you an account of this execution.

If you discover sockets, `socklab v6` will help you understand the role of different primitives, and how they should be used (what can be done, what is forbidden ...) If you have already known the sockets, working with `socklab v6` may be very useful to study their function or their limitations in some specific situations.

Version 4 of `socklab` was already available to manipulate IPv4 sockets. In the version 6 you will manipulate with IPv6 sockets.

Chapter 2

General presentation

2.1 Classification of commands

The commands of `socklab v6` are classified into three categories :

1. The *control* commands: allow creating, deleting or modifying their operation mode (manipulation options).

2. The *operating* commands: allow sending or receiving messages through the socket.

3. The *special* commands not directly related to the manipulation of sockets (for example: displaying the state of sockets created, displaying the list of available commands).

2.2 Operating Modes of `socklab v6`

`socklab v6` can be used in three different operation modes. Choosing one of these three modes determines the list of commands and control operations available for using `socklab v6` (in all three cases, the list of special commands is invariant) :

- In standard mode (default mode), a command is associated with each system allow to manipulate sockets (ex: primitives such as `socket()`, `bind()`, `send()`). By running a list of commands, you can see the results you will get if you run a program in C which is linked with list of primitive equivalent.

- In TCP mode, the range of available commands is directed to exclusive sockets TCP. Primitives unrelated with TCP are not represented: the commands available are less numerous than in standard mode. Moreover, the new commands are introduced to simplify the using of `socklab v6` (those commands have concatenation of several features).

- Similarly, in UDP mode, the list of available commands is directed to exclusive sockets with UDP.

Standard mode is normally used to study the primitive programming interface sockets, how they work and discover their limits. While TCP and UDP modes are normally used to study the respective protocols (in service or in operation, but in the last case, the joint use of a traffic analyzer like `snoop`, `tcpdump` or `ethereal` is however necessary).

2.3 Principle of use

Whatever the mode of operation used, the purpose of `socklab v6` is to provide an interface to create, operate and remove sockets. During one session with `socklab v6`, you can handle multiple sockets simultaneously. This means that between each command, `socklab v6` maintains a list of sockets that you have created. At any time you can select any previously created socket, and use it to send or receive data through the appropriate commands.

2.4 Launch `socklab v6`

`socklab v6` normally appears in the directory `/usr/local/bin/` of stations. Normally, this directory is automatically included in the `PATH` environment variable.

Once launched, `socklab v6` displays a banner presentation, followed by the prompt of the shell:

```
sparc1 # socklabv6
socklab v6 - laboratoire d'etude des sockets INTERNET
A.BOUKIOUD & Q.DUONG & R.NASRI - ENSIMAG - Juin 2011
------------------------------------------------------------------------
socklab>
```

By default, `socklab v6` operates in standard mode. To use UDP or TCP mode, you must specify the option respectively `udp` or `tcp` on the command line. The prompt `socklab v6` is then modified to indicate the mode used. For exemple :

```
sparc1 # socklabv6 tcp
socklab v6 - laboratoire d'etude des sockets INTERNET
A.BOUKIOUD & Q.DUONG & R.NASRI - ENSIMAG - Juin 2011
------------------------------------------------------------------------
socklabv6-TCP>
```

Chapter 3

Example of using `socklab v6`

In this section, we will illustrate the use of `socklab v6` in standard mode. Starting by launching the laboratory in this mode without running `socklab v6` option :

```
sparc1 # socklabv6
socklab v6 - laboratoire d'etude des sockets INTERNET
A.BOUKIOUD & Q.DUONG & R.NASRI - ENSIMAG - Juin 2011
------------------------------------------------------------------------
socklabv6>
```

In this mode, each primitive programming interface sockets has an associated command. Therefore, all commands have the same name as the primitive that they handle. For example, to create a socket, call the system's `socket()`. Ask the execution of the command `socket()` :

```
socklabv6> socket
Protocole ? [tcp/udp]:
```

The original `socket()` normally has three parameters: the area where the new socket must be created, its type and protocol is used in the given type. The laboratory is only used to study the Internet domain sockets. Both types are only represented (STREAM or DATAGRAM), and each correspond a single protocol (TCP or UDP, respectively). Therefore, `socklab v6` just asks the protocol you want to use with the new socket. Selecting UDP, `socklab v6` then has all the information necessary to call the original `socket()`. If the call does not fail, `socklab v6` returns the *identifier* of the socket that it has just created (this is the descriptor number as the original `socket()` returns). Through this identifier, you can now manipulate the socket :

```
socklab v6> socket
Protocole [tcp/udp] ?: udp
La socket est identifiee par l'identificateur 3

socklab v6>
```

In order not to make use of `socklab v6` too cumbersome, the names of commands can be given for short. Moreover, when a command has parameters (ex: the type of protocol for the command socket), they can be directly entered on the line. Thus, to create a new TCP socket, you can ask for different ways: `socket tcp`, `sock tcp`, `s tcp` or `s t` commands are equivalent. As the parameter is passed directly on the line, `socklab v6` makes you a prompt when it has executed the primitive `socket()`:

```
socklabv6> sock udp
La socket est identifiee par l'identificateur 4

socklabv6>
```

Since the beginning of this demonstration, two sockets have been created. To check, run the `status` command:

```
socklabv6> status
 Id      Proto       Adresse             Connexion           RWX ?
 ------------------------------------------------------------------------
 3       UDP         -                   -                   .W.
>4       UDP         -                   -                   .W.

socklabv6>
```

This command displays all the information attached to sockets that have been created (shortened form of the command status is `=`). For each socket displayed, `Id` is identified, `Proto` is the type of protocol used, `Adresse` is the address which was possibly affected (in the example above, this is not the case), `Connexion` indicates whether a connection is established, and RWX indicates that it is possible to make (`R` = read, `W` = write, `X` = eXception cf. 4.2.3). Note the sign `>` to the left of the second socket. This indicates what is the socket recently been created or manipulated.

You will now use the `bind` command, which uses the primitive of the same name to assign an address to a socket. In the Internet domain, a socket is identified by an *Internet address* and a *port number*.

Running the `bind` command without arguments, `socklab v6` will ask you to enter the parameters one by one, ie: the identifier of the socket address, internet address and port number to assign. For some arguments, `socklab v6` offers default values, as in the case of socket identifier : by default `socklab v6` offers the identifier of the last socket created or manipulated (the one that appears with the `>` sign with the `status` command). If `socklab v6` offers default values for arguments, they appear in parentheses. Just then press `Enter` to confirm.

In this example, we assign to the socket identifier 3, the station address `sparc1` : replace with the address or name of the station where you work. As for the port number, we choose arbitrarily the number `5000` :

```
socklabv6> bind
Id. socket (4) ?: 3
Host ?: sparc1
```

```
Port ?: 5000

socklabv6>
```

When a station is connected to several networks, it has several Internet addresses. The primitive `bind()` may accept a special website `in6addr_any` noted, which means any address of the machine (this is actually a generic address). The notation for use with the bind command to designate that particular address is the asterisk, `*`.

Regarding to the port, you can leave the operating system choosing a free port number (if the previous operation in the port `5000` was already used, the original `bind()` would have omited). To do this, specify the port number `0`, after running the initial `bind()`, `socklab v6` will display the port number chosen by the system. To illustrate these last two points, name the second UDP socket: :

```
socklab v6> bind 4 * 0
Le port 1043 a ete affecte a la socket

socklabv6> =
 Id      Proto       Adresse             Connexion           RWX ?
 ------------------------------------------------------------------------
 3       UDP         sparc1(5000)        -                   .W.
>4       UDP         *(1043)             -                   .W.
socklab v6>
```

You may notice that the two sockets now have an address. Until now you have used only two control commands, `socket` and `bind`, and a special command `status`. For finishing, you'll exchange a message between two UDP sockets created. The primitive for sending a message using the UDP protocol called `sendto ()`. The `sendto` command parameters are in order: the identifier of the socket to use, the machine destination, the destination port number on that machine and message to send. If the message consists several words, you must use quotation marks, if not all words will be considered as additional parameters. Using `sendto` command, send a message from the first socket created, Id 3, to the second `sparc1`, port `1043` :

```
socklabv6> sendto 3 sparc1 1043 "Voici un message"
16 octet(s) envoye(s)

socklabv6>
```

On the second UDP socket (marked with 4 in the example), ask the reception of 20 bytes per command `recvfrom`. This command has two parameters: the number of the socket on which the message must be read, and the number of bytes to read :

```
socklab v6> recvfrom 4 20
Un message de 16 octet(s) a ete recu de sparc1 (5000).
Message=<Voici un message>

socklab v6>
```

Now what happens when `socklab v6` has a problem while running a command. You try using the `accept` command on the last UDP socket created. This command is reserved for the TCP protocol to retrieve a connection request on a socket passive. It makes no sense for UDP :

```
socklab v6> accept 4
accept(): Operation not supported on transport endpoint

socklab v6>
```

When calling a primitive result in a failure, a message describing the name of the primitive called and the error found is still displayed in reverse video (this message is accompanied by a beep).

Note that using `socklab v6` was shown on a single machine, but you can obviously use the lab on two different machines connected by networks, and implement communications between these two laboratories through socket TCP or UDP . At any time, you can request a list of available commands, type `help` or `?`:

```
socklabv6> help
LISTE DES COMMANDES DISPONIBLES:
    s socket            r read              w write           ? help
    b bind              v recv              d send            = status
    l listen            f recvfrom          t sendto          q quit
    a accept            x exit
    c connect           <id> select. sock.
    k close
    h shutdown
    o options

socklabv6>
```

Each command appears with a letter that might be used in its place as minimum abbreviation.

Chapter 4

`socklab v6`'s Reference Guide

4.1 Utilization

The laboratory of sockets study starts by typing `socklab v6` from a shell. Specifying parameter `udp` or `tcp`, you can guide the manipulation of sockets to UDP or TCP, respectively.

Once the lab started, you can access to a shell that lets you create, select, delete or manipulate sockets.

The laboratory maintains a permanent table which stored the list of sockets that you have created.

Most commands require one or more parameters: You can specify as arguments on the command line, otherwise they will be automatically applied when executing the command.

Some commands also have options such as the commands `send` or `recv`: these options can be specified on the command line. If you omit them, they will not be required when running the command (options are words that always start with a dash `-`).

You can write the shorten names of commands, parameters or options, provided that the interpreter can identify them without ambiguity. For example, you can type `so` instead of the `socket` command. In contrast, you can not type `se` instead of `send` command because the `sendto` commence also by `se`. However, each command is associated with a single letter which can be used in place of the full name.

At any time, you can interrupt the progress of a command by pressing `Ctrl-C`: you will return automatically to the shell of the laboratory. We will now describe separately all commands supported by the interpreter `socklab v6`. For each we show the first letter can be used as a shortcut, then its possible parameters and options.

4.2 Special commands

Special orders are those that do not apply to a particular socket. They are always available, whatever the mode of operation of the laboratory.

4.2.1 `Q|X - quit|exit`

Description : These two commands are equivalent: they can leave the lab to return to the shell. Confirmation is still required (all sockets created will be deleted upon termination of `socklab v6`).

4.2.2 `?  - help` *`<cmd> ...`*

Description : without parameters, this command displays a list of all available commands with the letter which can be used as a shortcut. If command names are passed as a parameter, the command displays only their syntax.

Parameter :

`<cmd>` Name of a command whose syntax must be displayed.

4.2.3 `= - status`

Description : Displaying the status of all sockets created. The last socket created or manipulated is indicated by the mark `>` at the beginning of the line that describes it. For each socket command displays :

1. identifier ;
2. protocol used: TCP or UDP;
3. local address if it has one (with its port number);
4. address of the remote socket if a connection was made;
5. feasible operations:

   `R` (read) reading data or recovery a connexion;
   `W` (write) writing data;
   `X` (exception) reading exception.

4.3 Standard Mode Commands

In this mode, all commands correspond to a particular primitive programming interface sockets (the mode of operation of the laboratory by default). The name of each command is one of the primitive manipulated.

In `socket`, all commands have first identifier parameter *`<id>`* to which they apply. For this parameter you can specify a point (`.`) to indicate that you want to use the latest socket created or manipulated.

4.3.1 `S - socket [tcp|udp]`

Description : creating a socket in the Internet domaine, type STREAM or DATAGRAM, depending on the value of the parameter.

Parameter :

`tcp` creating a socket of type STREAM (using TCP) ;

`udp` creating a socket of type DATAGRAM (using UDP).

4.3.2 B - bind *<id> <host> <port>*

Description : assigning an address to the socket `<id>`.

Parameters :

<host> IPv6 Internet address assigned to the socket. If the machine you're working on is connected to multiple networks, it has several addresses. This setting allows you to specify one. You can type directly IPv6 address (*dot notation* : `xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx`) or you can give a symbolic name in the file `/etc/hosts` (`.` is a shorthand for the official name of the machine where you work).

You can specify the particular address `in6addr_any` with asterisk `*`. Address `in6addr_any` is generally used for passive TCP sockets, in order to receive connection requests on any machine address.

<port> port number of the socket. You can directly type the port number or you can give the name of a service in the file `/etc/services`. By providing the value `0`, the system automatically selects an available port number.

4.3.3 L - listen *<id> <nb>*

Description : listening setting of the socket *<id>* (only TCP socket).

Parameter :

<nb> number of connections can be maintained until the last ones are recovered by the command `accept` (default, `nb = 5`).

4.3.4 A - accept *<id>*

Description : awaiting or recovery a connection on the socket *<id>* (only TCP socket).

4.3.5 C - connect *<id> <host> <port>*

Description : connection socket *<id>* on a remote socket. If the connection has been established, a new socket will be created.

Parameters :

<host> name or IPv6 address of the remote machine (cf. `bind`) ;

<port> port number or service name on the remote machine.

4.3.6 K - close *<id>*

Description : closing the socket *<id>*.

4.3.7 H - shutdown <*id*> [in|out|both]

Description : closing the previous established connection on a socket <*id*>.

Parameter :

`in` closing input only ;

`out` closing output only ;

`both` closing in both directions.

4.3.8 W - write <*id*> <*msg*>

Description : writing a message in the previous established connection on a socket <*id*>.

Parameter :

<*msg*> message to send. If you want to send several words, you must limit them in quotes. To send a specific number of bytes, regardless of message content, use the notation `#nnn` where `nnn` is the number of bytes to send (the message is composed by asterisks `*`).

4.3.9 R - read <*id*> <*nb*>

Description : reading a message in the previous established connection on a socket <*id*>.

Parameter :

<*nb*> number of bytes to read (100 by default).

4.3.10 D - send <*id*> <*msg*> [-oob] [-dontroute] [-loop]

Description : writing a message in the previous established connection on a socket <*id*>. This command differs from `write` only the options it offers.

Parameters :

<*msg*> message to send (cf. `write`).

Options :

`-oob` option *Out Of Band*, to send the message in mode *urgent* ;

`-dontroute` option *Don't Route*, for the message can not cross a router ; in other words, he can not go outside the network on which the machine is connected.

`-loop` This option does not appear in the standard primitive `send()`. In the specifying, `socklab v6` sends the message until an error intervene or you press `Ctrl-C`.

4.3.11 V - recv <id> <nb> [-oob] [-peek]

Description : reading a message in the previous established connection on a socket <id>. This command differs from `read` only by the options it offers.

Parameter :

<nb> number of bytes to read.

Options :

`-oob` option *Out Of Band*, to read the message in mode *urgent* ;

`-peek` option *Peek*, to read the data while in the receive buffer ; This option allows you reread the same data.

4.3.12 T - sendto <id> <host> <port> <mesg> [-oob] [-dontroute] [-loop]

Description : Sending a message to a specific destination socket. The socket <id> is used as the source of the message (UDP socket only)..

Parameters :

<host> IPv6 address of the destination machine.

<port> number of the socket on the destination machine.

Options :

`-oob` option *Out Of Band* (cf. `send`) ;

`-dontroute` option *Don't Route* (cf. `send`) ;

`-loop` option *Loop* (cf. `send`).

4.3.13 F - recvfrom <id> <nb> [-oob] [-peek]

Description : receiving on the socket <id>, a message sent by a remote socket (UDP socket only).

Parameter :

<nb> number of bytes to read.

Options :

`-oob` option *Out of Band* (cf. `recv`)

`-peek` option *Peek* (cf. `recv`)

4.3.14 `O - options <id> <option> ...`

Description : viewing and editing options of the socket *<id>*. Used without parameters, this command reviews all options for the socket : they are displayed one by one with their current value. If you specify one or more specific options for each parameter, the command only deals with them.

For not changing the value of an option, type directly on `Enter` ; The next option is displayed.

The majority of options are boolean : you must respond by `o` or `n` to modify them.

The option `asynchrone` allows to be notified when a message or a connection request arrives on the socket.

The option `nodelay` allows you to make non-blocking primitives (by default, a `read` is blocked when no message is ready).

All other options are those that can be accessed or modified by the primitive `getsockopt()` and `setsockopt()` (However, all the options normally available are not shown in `socklab v6`).

4.4 Commands in TCP mode

TCP mode is the mode obtained by running the lab with the parameter `tcp`. The objective of this mode is to study the TCP protocol, and not the sockets themselves. The available commands are less numerous but simple to use because they generally carry the sequence of several primitives manipulating sockets.

In `passive` and `connect`, all the commands have first identifier parameter *<id>* of the socket to which they apply. For this parameter you can specify a point (`.`) to indicate that you want to use the latest socket created or manipulated.

4.4.1 `P - passive`

Description : creating a socket passive, ready to receive connection requests. The address `in6addr_any` is automatically assigned, and a free port number is chosen by the system (equivalent to `socket tcp`, `bind . * 0` and `listen . 5` in standard mode).

4.4.2 `A - accept <id>`

Description : same as `accept` in standard mode.

4.4.3 `C - connect <host> <port>`

Description : creating an active socket and then attempt to connect to a remote socket. The address `in6addr_any` is automatically assigned, and a free port number is chosen by the system (equivalent to `socket tcp`, `bind . * 0`, `connect . <host> <port>` in standard mode).

Parameter :

<host> name or IPv6 address of the remote machine ;

<port> port number of the socket on the remote machine.

4.4.4 K - close *<id>*

Description : same as `close` in standard mode.

4.4.5 H - shutdown *<id>* [in|out|both]

Description : same as `shutdown` in standard mode.

4.4.6 W - write *<id>* *<msg>*

Description : same as `write` in standard mode.

4.4.7 R - read *<id>* *<nb>*

Description : same as `read` in standard mode.

4.4.8 D - usend <id> <msg>

Description : Writing an urgent message to the previous established connection on a socket *<id>* (equivalent to `send` *<id>* *<msg>* `-oob` in standard mode).

Parameter :

<msg> message to send (cf. `write`)

4.4.9 V - urecv *<id>* *<nb>*

Description : reading an urgent message to the previous established connection on a socket *<id>* (equivalent to `recv .` *<nb>* `-oob` in standard mode).‘

Parameter :

<nb> number of bytes to read.

4.4.10 O - options *<id>* *<option>* ...

Description : same as `options` in standard mode.

4.5 Commands in UDP mode

UDP mode is the mode obtained by running the lab with the parameter `udp`. The objective of this mode is to study the UDP protocol, not the sockets themselves.

Apart from socket, all the commands have first identifier parameter *<id>* to which they apply. For this parameter you can specify a point (`.`) to indicate that you want to use the latest socket created or manipulated.

4.5.1 S - socket

Description : Creating a UDP socket. The address `in6addr_any` is automatically assigned, and a free port number is chosen by the system (equivalent to `socket udp` and `bind . * 0` in standard mode).

4.5.2 K - close *<id>*

Description : same as `close` in standard mode.

4.5.3 T - sendto *<id> <host> <port> <msg>*

Description : same as `sendto` in standard mode, fewer options.

4.5.4 F - recvfrom *<id> <nb>*

Description : same as `recvfrom` in standard mode, fewer options.

4.5.5 O - options *<id> <options> ...*

Description : same as `options` in standard mode.

Index

accept, 11, 14, 17

bind, 9, 14

close, 14, 18, 19
connect, 14, 17

exit, 13

help, 11, 13

listen, 14

options, 17–19

passive, 17

quit, 13

read, 15, 17, 18
recv, 16
recvfrom, 10, 16, 19

send, 15
sendto, 10, 16, 19
shutdown, 15, 18
socket, 13, 19
status, 9, 13

urecv, 18
usend, 18

write, 15, 18